Performance and Consistency in Learning to Program

Ahadi, Alireza

ACM
2017-01-31

Ahadi , A , Lister , R , Lal , S , Leinonen , J & Hellas , A 2017 , Performance and Consistency in Learning to Program . in Proceedings of the Nineteenth Australasian Computing Education Conference . ACM , New York, NY , pp. 11-16 , Australasian Computing Education Conference , Geelong , Australia , 31/01/2017 . https://doi.org/10.1145/3013499.3013503

http://hdl.handle.net/10138/316010
https://doi.org/10.1145/3013499.3013503

acceptedVersion

Performance and Consistency in Learning to Program

Alireza Ahadi and Raymond Lister
University of Technology, Sydney
Australia
alireza.ahadi@uts.edu.au
raymond.lister@uts.edu.au

Shahil Lal
University of Sydney, Australia
shahil.lal7@gmail.com

Juho Leinonen and Arto Hellas
Department of Computer Science
University of Helsinki
Finland
juho.leinonen@helsinki.fi
arto.hellas@cs.helsinki.fi

ABSTRACT

Performance and consistency play a large role in learning. Decreasing the effort that one invests into course work may have short-term benefits such as reduced stress. However, as courses progress, neglected work accumulates and may cause challenges with learning the course content at hand.

In this work, we analyze students' performance and consistency with programming assignments in an introductory programming course. We study how performance, when measured through progress in course assignments, evolves throughout the course, study weekly fluctuations in students' work consistency, and contrast this with students' performance in the course final exam.

Our results indicate that whilst fluctuations in students' weekly performance do not distinguish poor performing students from well performing students with a high accuracy, more accurate results can be achieved when focusing on the performance of students on individual assignments which could be used for identifying struggling students who are at risk of dropping out of their studies.

Keywords

source code snapshot analysis; educational data mining; CS1

1. INTRODUCTION

Researchers have sought to determine whether factors such as gender, age, high-school performance, ability to reason, and the performance in various aptitude tests correlate with the ability to create computer programs [6]. As many of these factors are static and only rarely account for *what the students do* while programming, studies that analyze the learning process have started to emerge [11].

In these studies, researchers study features extracted from programming process recordings at various granularity [1, 5, 12, 24]. Process data has been collected also in other contexts, e.g. during in-class peer instruction [15]. With such research, in the future, *data analytics* and support tools can be regularly applied to provide instructors with greater insight into what is actually occurring in the classroom, opening up new opportunities for identifying individual student needs, providing targeted activities to students at the ends of the learner spectrum, and personalizing the learning process [11].

In our work, we are interested in how students' performance evolves during the course and how their performance and consistency contribute to the course outcomes. More specifically, we study how students' performance, measured through correctness of snapshots taken from students' programming process during an introductory programming course, evolves over time and analyze whether students consistently perform on the same level. The analysis is based on observing students' average performance, and seeks to determine whether the weekly performance fluctuates due to some unobserved variables. Furthermore, we also study students' performance using non-parametric and parametric clustering approaches with the goal of detecting those students who could benefit from a teaching intervention.

This article is organized as follows. In the next Section, we discuss related work. In Section 3, we describe the research questions and data, and in Section 4, we outline the methodology and results. The results are further discussed in Section 5 and drawn together in Section 6.

2. RELATED WORK

2.1 Predictors for Success

Traditional predictors for students' success include past academic performance [2], previous exposure to programming [7], and demographic factors such as age and gender [25]. In addition, variables such as students' expectations for course and for their grade correlates with the final course outcomes [16]. However, as pointed out by Watson et al. [23], many of the traditional predictors are sensitive to the teaching context, and generalize relatively poorly.

There are studies that indicate that there is no significant correlation between gender and programming course outcomes [3, 21, 25], as well as studies that indicate that the gender may explain some of the introductory programming course outcomes [2]. Similarly, when considering past mathematics performance, some studies indicate no significant correlation between programming course outcomes and mathematics outcomes [23, 25], while others have found referential correlations [19].

Even the connection between introductory programming course outcomes and past exposure to programming is controversial. Whilst a number of studies have reported that past programming experience helps when learning to program [4, 7, 26], contradictory results also exist. For example,

ACE '17, January 31-February 03, 2017, Geelong, VIC, Australia

DOI: http://dx.doi.org/10.1145/3013499.3013503

Bergin and Reilly found that students with no previous programming experience had a marginally higher mean overall score in introductory programming [2]. Watson et al. [23] also found that while students with past programming experience had significantly higher overall course points than those with no previous programming experience, programming experience in years had a weak but statistically insignificant negative correlation with course points.

A more recent approach to identifying students who may or may not succeed is related to the use of data that is recorded from the environment in which the students are learning to program. We focus on such studies next.

2.2 Work Patterns and Effort

In-class behavior has been explored by Porter et al. [15] who observed that the proportion of correct clicker answers early in a course was strongly correlated with final course outcomes. Similarly, interaction with web-based programming environments has been studied [18]; Spacco et al. noticed statistically significant but weak relationships between the final exam score and students' effort.

Sequential captures of programming states have been used to analyze students' working patterns [11]. Jadud found a high correlation with the course exam score and the students' ability to create and fix errors [12], and observed that the less errors a student makes, and the better she solves them, the higher her grade tends to be [12].

The approach proposed by Jadud has been extended by Watson et al. [24] to include the amount of time that students spend on programming assignments [24]. More recently, a similar approach that analyzes subsequent source code snapshots was also proposed by Carter et al. [5], who analyzed the errors that students encounter in more detail.

Programming states have been used to elicit finer information from the programming process. For example, Vihavainen et al. [22] analyzed the approaches that novices take when building their very first computer programs. They observed four distinctive patterns, which were related to typing the program in a linear fashion, copy-pasting code from elsewhere, using auto-complete features from the programming environment, and using programming environment shortcuts. Similar work was conducted by Heinonen et al. [8], who analyzed problem solving patterns of two populations in an introductory programming course; those who failed and those who passed. In the study, Heinonen et al. noticed that a number of the students who failed had a *programming by incident* -approach, where they sought a solution through a seemingly random process – a behavior that has also observed in the past (see e.g. [17]).

One stream of research merges snapshots into states, and analyzes students' progress through those states. Piech et al. [14] clustered students' approaches to solving a few programming tasks, and found that the solution patterns are indicative of course midterm scores. Similarly, Hosseini et al. [9] analyzed how students' progress towards correct solutions in a course, and noticed that some students were more inclined to build their code step by step from scratch, while others started from larger quantities of code, and progressed towards a solution through reducing and altering the code.

Students have also been grouped based on features describing how they work on programming assignments (lines changed, added, removed, and so on), followed by an evaluation of how these features change over a number of assignments [27]. Overall, students in different groups may differ in the way how they benefit from help [27].

Our research is closely related to the work by Worsley et al. [27] and Hosseini et al. [9]. However, whilst both have focused more on change sizes and other similar metrics, our focus is on changes in correctness of code. We also evaluate new methodologies and visualizations for the task at hand, and study the students' performance across a course instead of over a smaller set of assignments.

3. RESEARCH DESIGN

3.1 Research Questions

Our research questions are as follows:

- **RQ1: How does students' performance evolve during a programming course?**
- **RQ2: How does the performance change across course weeks? That is, how consistently do students perform during the course?**
- **RQ3: To what extent can students' consistency be used to predict students' course outcomes?**

With research question one, we aim to both validate previous results by Spacco et al. [18] in a new context, and to extend the work by analyzing students' performance changes from week to week. These changes are discussed in the light of the content taught in the course. With research question number two, we seek to determine whether working consistency – i.e. does student perform similarly over the course – affects course outcomes, and with the third research question, we revisit multiple works where authors have sought to determine those at risk of failing the introductory programming course.

3.2 Context and Data

The data for the study comes from a six-week Java introductory programming course organized at University of Helsinki during Spring 2014. One of the authors of this article is responsible for organizing the course under study. In the course, half of the grade comes from completing programming assignments, and the rest of the grade comes from a written exam, where students are expected to answer both essay questions as well as programming questions. The course has a set limit for the pass rate: at least half of the overall points as well as half of the exam points need to be attained to pass, and the highest grade is attained with over 90% of course points.

The course was based on a blended online textbook, had a single two-hour lecture, and tens of hours of weekly support in open labs. The course provides students a view to both procedural and object-oriented programming. Course assignments start easy and small, helping students to first focus on individual constructs before they are combined with others. After the students have become familiar with the basic tools and commands, variables and conditional statements are introduced, followed by looping. Students start constructing their own methods during the second week in the course, and start working with lists during the week three. Principles of object-oriented programming are introduced during the latter parts of the third week of the course, and students start building their own objects during the fourth week. Overall, during the course, the students slowly proceed towards more complex assignments and topics.

For the purposes of this study, students were asked to install a software component that records source code snap-

shots with metadata on assignment-specific correctness. Overall, 89 students agreed to provide their data for the purposes of our study.

3.3 Measuring Performance and Consistency

Throughout this work, students' performance and consistency is measured through their progress and work on the course assignments. For each assignment in the course, there are automatic unit tests, which are used to both support the students as they work on the assignment, and to provide information on the degree of correctness on the assignment. As the students work on the assignments, data is gathered for analysis. Finally, at the end of the course, the students take an exam. In this work, when seeking to predict course outcomes, we focus on predicting the outcomes of the exam.

Students' *average performance* in an assignment means the average correctness of snapshots that have been recorded when the students have either saved, run, tested, or submitted the code that they are working on. The average is calculated for each assignment for all students who worked on the assignment. Average performance provides a measure of the degree to which students' struggled on an assignment and indicates overall performance throughout the course.

Students' *consistency* is measured through whether they perform on the same level throughout the course. More specifically, for each week, we place students' into weekly performance quantiles and measure whether they perform on the same level (i.e. stay in the same quantile), or whether their performance level fluctuates (i.e. changes over the weeks). The consistency is used both as a way to measure the weekly struggle as well as the effort that the student invests into the assignment at a specific state; if a student is always in the upper quantile, most of the effort is invested in complete or nearly complete assignments.

4. METHODOLOGY AND RESULTS

4.1 Students' Performance over the Course

First, we performed statistical analysis on the assignment correctness data gathered from the students' programming process. Average performance (Figure 1) was in focus to analyze how students' performance evolves during the programming course.

A first order (i.e. linear) regression analysis of students' performance throughout the course shows that average performance declines throughout the course. Second order regression shows that the average performance of students increases until the middle of the course, and then declines towards the end.

Overall, students performance decreases throughout the course. This observation is partially explainable by the incremental nature of programming. Students proceed towards more complex assignments and topics and new content is added continuously. At the same time, the results show that the peak average performance is reached at the middle of the course when students start to work on object-oriented programming.

4.2 Students' Performance over Course Weeks

After analyzing the overall trend, we study whether students perform consistently throughout the course, or whether their performance varies. This analysis was performed using weekly average performance quantiles.

In Figure 2, students are placed into four categories based on their average performance throughout each week. If the average performance of a student during the whole week was

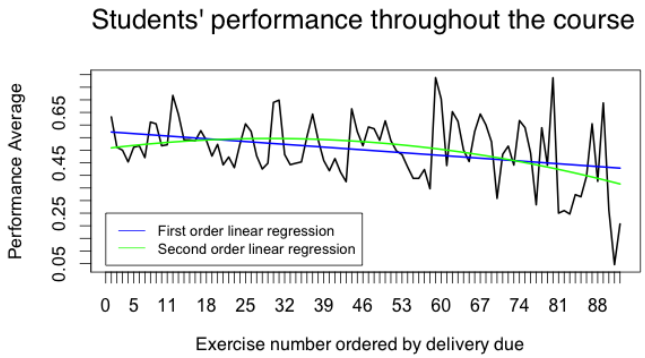

Figure 1: First order and second order regression plot of the mean of performance per exercise.

over 75%, she was in the high-performing quantile (green in Fig. 2), whilst if the student's average performance was less than or equal to 25%, she was in the low-performing quantile (red in Fig. 2). As the semester progresses, there is a noticeable decline in the number of students who belong to the high-performing quantile. At the end of the course, nearly 50% of the students are in the lowest performing quantile, and only a very small number of students remained in the high-performing quantile.

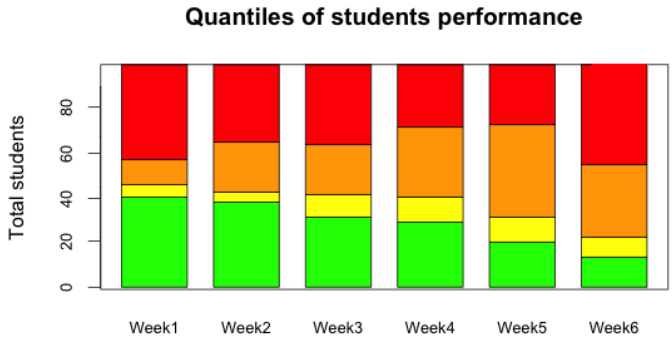

Figure 2: Frequency of students in different quantiles per week in the course. The colors green, yellow, orange and red represent 1st, 2nd, 3rd and 4th quantiles respectively.

We then set out to analyze the extent to which a student's performance during one week indicates performance during the next week. To visualize this, a state transition diagram for the student's performance transition from one week to the next week was constructed. These transitions are calculated using the previous weekly performance quantiles, where a student's average performance in all weekly assignments could fall in one of the four quantiles.

When analyzing the transitions (Figure 3), the majority of students show neither progression or retrogression. That is, a large part of the students perform somewhat consistently between any two weeks of the course. At the same time, early in the course, majority of the students fell either into the upper quantile or the lower quantile, whilst at the end of the course, majority of the participants had a low performance average.

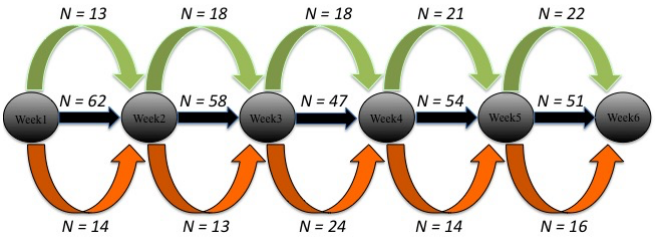

Figure 3: State transition diagram of students' performance throughout the course.

4.3 Consistency and Course Outcomes

Using students' consistency to predict course outcomes is performed in two parts. We first use the consistency over the weeks for the course outcome prediction, and then, in the next subsection, we delve into assignment-specific performance and course outcomes.

The analysis was performed using t-Distributed Stochastic Neighbor Embedding (t-SNE) clustering [20], which is a technique for dimensionality reduction suitable for high-dimension datasets. For the analysis, three separate datasets were used. The first dataset contained students' average performance per week, the second dataset contained students' weekly performance quantiles, and the third contained students' state transitions between performance quantiles.

For each dataset, an analysis of a limited range of weeks was performed to evaluate whether the clustering method could be used as an early indicator of performance, as well as to identify weeks where students' performance varied the most. Kullback-Leibler divergence was used as a measure for the difference between the clustering outcome and the course outcome for each dataset. Values of the analysis are summarized in Table 1. Here, the consistency dataset (dataset 3) has entries only in the rows with more data as transitions between any two weeks did not yield meaningful results.

Table 1: t-SNE Kullback-Leibler divergences of three different datasets.

period	dataset 1	dataset 2	dataset 3
Weeks one and two	0.25	0.36	NA
Weeks two and three	0.27	0.36	NA
Weeks three and four	0.2	0.25	NA
Weeks four and five	0.23	0.29	NA
Weeks five and six	0.18	0.28	NA
Weeks one to three	0.25	0.33	0.38
Weeks two to four	0.24	0.35	0.35
Weeks three to five	0.26	0.24	0.34
Weeks four to six	0.28	0.25	0.35

The Kullback-Leibler divergence of the students' state transition data (dataset 3) is higher when compared to both weekly performance (dataset 1) and weekly quantiles (dataset 2). That is, the average weekly performance performs better as an indicator of final exam result than the transitions between the performance quantiles.

The result of t-SNE clustering on students' average performance per week is shown in Figure 4 – here, we have used the data from weeks four, five and six with Kullback-Leibler divergence 0.28. T-SNE does not require an initial value for the number of desired clusters, that is, it is a non-parametric approach. One of the major benefits of non-parametric models is that the parameters are determined by the data, not the chosen model.

As could be seen in Figure 4, t-SNE was able to cluster students based on their performance into meaningful clusters – few indicating mostly passing students, one with mostly dropouts, and one with mixed data. The labels in the figure depict those students who pass the subject (`Y`), those who failed the subject (`N`), and those who drop out from the course (`Z`).

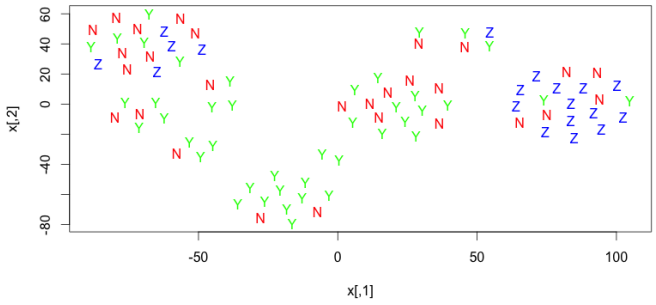

Figure 4: t-SNE based visualization of students final course outcomes using data from weeks four, five and six of students' average performance.

4.4 Assignment-specific Performance and Course Outcomes

Thus far, students were clustered based on their weekly performance. We observed that performance fluctuations between weeks is not as indicative of performance in the final exam as the average weekly performance. We now move from analyzing weekly performance to analyzing assignment-specific performance and its relationship with course outcomes. As the number of assignments is high, dimensionality reduction over the assignments using principal component analysis (PCA) [10] was performed.

Whilst performing PCA on the assignment data, we also performed k-means clustering [13] to identify a meaningful amount of groups into which the data could be split into. Upon analysis of the k-means clustering results over k between 1 and 10, we observe that the projection of our data could be best explained in a two-dimensional space with two principal components. We then used those two principal components to divide the students into two groups. As a result, 78% of students who failed or dropped the subject are placed into the correct cluster.

The same analysis was then performed on three datasets similar to the previously used datasets: (a) the quartile of each student's performance on each exercise; (b) the state transition data based on the score of each exercise where the student's performance from one week to the next week could be either progress, retrogress or no change; and (c) the state transition data based on the quartile of the obtained score of the student per exercise where the student's performance from one week to the week after could be either progress, retrogress or no change.

When performing clustering on the three datasets, we observed that the used k-means clustering does not show a

good separation of successful and unsuccessful students. The result of our analysis suggests that at-risk students could best be predicted by the scores or quantile of exercises (i.e. datasets a and b, being able to correctly identify 78.72% and 76.60% of the at-risk students respectively), compared to the quartile state of student's performance or performance transition between exercises. However, the predictive information stored in state transitions seem to be more effective in identifying students who perform well in the final exam of the course (identifying 66.67% of well-performing students correctly).

5. DISCUSSION

5.1 Overall performance in the course

Overall, in line with previous studies, we observed that students' performance decreases during the course. This is in line with both the general knowledge in introductory programming as well as the literature where students' performance in the online programming environment has been analyzed [18]. However, when contrasting our results to those in Spacco et al. [18], students in our context are working in a traditional programming environment, and the majority of the course assignments expect that the students implement more than a single function – thus, generalizing the previous results to a new context.

Upon analysis of the average performance, in Fig. 1, we observed through second order regression that the performance first increases, and then decreases. We observe that students in the course perform reasonably well until the introduction of object-oriented programming, after which the assignments become more complex.

Overall, course topics build on each other, and learning each new topic depends on whether the student understands the previous topic to a sufficient detail. If a student struggles on a topic in the course and fails to grasp it, he or she will likely struggle with upcoming topics as well. Second, as programming courses progress, it is typical that the complexity of the programming assignments increases.

5.2 Changes in performance

When analyzing the students performance transitions in the course, we observe that for the majority of the students, the performance stays nearly the same. When analyzing performance through quantiles, nearly none of the students perform at a level where their solutions would, on average be over 75% correct. What we likely also observe here is that as the course progresses, students become less likely to tinker with their solutions after they have finished the solution, leading to a smaller amount of snapshots in states with high correctness.

At the same time, as the assignments are more complex, starting an assignment becomes more challenging, and approaching a correct solution takes more steps than before. This means that as students struggle, they perform worse in the light of our metric. At the same time, struggling does not necessarily indicate poor *learning*, only that the assignments are more challenging.

We observe that fluctuations in average performance does not transfer well to fluctuations in course performance. Our results are in line with those in [27], indicating that students' performance does not drastically change during the semester.

5.3 Clustering and at-risk students

When comparing the clustering results to the results previously presented in the literature, the outcomes are mediocre, and we do not observe a good distinction between the students who perform well and those who do not. Recent studies that have focused on selecting features that best determine students who are at risk of dropping out have reported high correlations. For example, a recent result from [1] reported $MCC = 0.78$, when using only the first week of data in a course and seeking to predict whether students fall under or over the class median, a significantly better result than that from our study. At the same time, the data, features and methodology was different to ours.

One of the reasons for the lack of performance in the clustering is that distinguishing between those students who pass and those who fail is not trivial. There is a clear separation between clusters of students who are far from the fail/pass -border, while the course outcome for a student who receives 49/100 (i.e. a fail in our system) and a student who receives 50/100 (i.e. a pass in our system) is large in terms of grading, there is very little difference in how they fared in the course and in the exam.

5.4 Limitations of the study

Our study has several limitations. First, the sample size in our dataset was only 89, which is relatively small for clustering and PCA. Second, the exam results come from a pen-and-paper exam, which does not fully reflect the activities that students perform during the course. It is possible that students handle programming tasks well, but are, for example, not able to properly explain what they are doing using pen and paper. On the other hand, students may get lower course marks because they are not able to complete the more complex assignments, but that does not necessarily mean that a student did not learn the topic and would not be able to do well in an exam.

Third, when analyzing performance transitions, we limited our work to quartiles. It is possible that better results would have been achieved with different settings such as a different interval for the quartiles.

Finally, we do not know if students do their best on the assignments as it is possible that some students only seek to pass the course. Additional details should be incorporated to the clustering to take such students into account.

6. CONCLUSIONS

In this work we analyzed students performance and consistency in an introductory programming course. For the analysis, we used sequential source code snapshots that described students progress with the programming assignments. We investigated course-level changes in students' performance, analyzed whether the performance varied over different weeks, and evaluated clustering to identify students at risk.

Overall, to answer research question one, *"How does students' performance evolve during a programming course?"*, we observe that students' performance declines throughout the course. This result is in line with previous studies (see e.g. [18]), and is explainable through the incremental nature of programming. To proceed in a course, the student has to learn the topics from the previous week.

Whilst the overall performance declined over the course, at the same time, we observe that the majority of the students have a constant performance from week to week. That is, to answer the research question two, *"How does the performance change across course weeks?"*, the performance

mostly stays the same. This means that the majority of the students perform similarly throughout the course, and that the overall decline in the course performance is likely explainable by the increasing complexity of the course assignments – a topic for future study.

When evaluating this performance as a predictor of final course outcomes, we noticed that fluctuation in course performance across weeks is a relatively poor metric for final course performance. The most accurate clustering was achieved using average performance. To answer the research question three, *"To what extent can students' performance be used to identify those that may drop out from a course?"*, we posit that one could use the clusters built from average performance. However, at the same time, one should seek to verify that there is a sufficient number of study samples, and that the measured outcome depicts the students' working process.

7. REFERENCES

[1] A. Ahadi, R. Lister, H. Haapala, and A. Vihavainen. Exploring machine learning methods to automatically identify students in need of assistance. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ICER '15, pages 121–130, New York, NY, USA, 2015. ACM.

[2] S. Bergin and R. Reilly. Programming: factors that influence success. *ACM SIGCSE Bulletin*, 37(1):411–415, 2005.

[3] P. Byrne and G. Lyons. The effect of student attributes on success in programming. In *ACM SIGCSE Bulletin*, volume 33, pages 49–52. ACM, 2001.

[4] B. Cantwell Wilson and S. Shrock. Contributing to success in an introductory computer science course: a study of twelve factors. In *ACM SIGCSE Bulletin*, volume 33, pages 184–188. ACM, 2001.

[5] A. S. Carter, C. D. Hundhausen, and O. Adesope. The normalized programming state model: Predicting student performance in computing courses based on programming behavior. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ICER '15, pages 141–150, New York, NY, USA, 2015. ACM.

[6] G. E. Evans and M. G. Simkin. What best predicts computer proficiency? *Comm. of the ACM*, 32(11):1322–1327, 1989.

[7] D. Hagan and S. Markham. Does it help to have some programming experience before beginning a computing degree program? *ACM SIGCSE Bulletin*, 32(3):25–28, 2000.

[8] K. Heinonen, K. Hirvikoski, M. Luukkainen, and A. Vihavainen. Using codebrowser to seek differences between novice programmers. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 229–234, New York, NY, USA, 2014. ACM.

[9] R. Hosseini, A. Vihavainen, and P. Brusilovsky. Exploring problem solving paths in a Java programming course. In *Proceedings of the 25th Workshop of the Psychology of Programming Interest Group*, 2014.

[10] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[11] P. Ihantola, A. Vihavainen, A. Ahadi, M. Butler, J. Börstler, S. H. Edwards, E. Isohanni, A. Korhonen, A. Petersen, K. Rivers, M. A. Rubio, J. Sheard, B. Skupas, J. Spacco, C. Szabo, and D. Toll. Educational data mining and learning analytics in programming: Literature review and case studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports*, ITICSE-WGR '15, pages 41–63, New York, NY, USA, 2015. ACM.

[12] M. C. Jadud. Methods and tools for exploring novice compilation behaviour. In *Proceedings of the Second International Workshop on Computing Education Research*, ICER '06, pages 73–84, New York, NY, USA, 2006. ACM.

[13] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[14] C. Piech, M. Sahami, D. Koller, S. Cooper, and P. Blikstein. Modeling how students learn to program. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 153–160, New York, NY, USA, 2012. ACM.

[15] L. Porter, D. Zingaro, and R. Lister. Predicting student success using fine grain clicker data. In *Proceedings of the tenth annual conference on International computing education research*, pages 51–58. ACM, 2014.

[16] N. Rountree, J. Rountree, and A. Robins. Predictors of success and failure in a cs1 course. *ACM SIGCSE Bulletin*, 34(4):121–124, 2002.

[17] J. Spacco. *Marmoset: a programming project assignment framework to improve the feedback cycle for students, faculty and researchers*. PhD thesis, 2006.

[18] J. Spacco, P. Denny, B. Richards, D. Babcock, D. Hovemeyer, J. Moscola, and R. Duvall. Analyzing student work patterns using programming exercise data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 18–23, New York, NY, USA, 2015. ACM.

[19] M. V. Stein. Mathematical preparation as a basis for success in CS-II. *Journal of Computing Sciences in Colleges*, 17(4):28–38, 2002.

[20] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

[21] P. R. Ventura Jr. Identifying predictors of success for an objects-first CS1. 2005.

[22] A. Vihavainen, J. Helminen, and P. Ihantola. How novices tackle their first lines of code in an ide: analysis of programming session traces. In *Proceedings of the 14th Koli Calling International Conference on Computing Education Research*, pages 109–116. ACM, 2014.

[23] C. Watson, F. W. Li, and J. L. Godwin. No tests required: Comparing traditional and dynamic predictors of programming success. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 469–474, New York, NY, USA, 2014. ACM.

[24] C. Watson, F. W. B. Li, and J. L. Godwin. Predicting performance in an introductory programming course by logging and analyzing student programming behavior. In *Proceedings of the 2013 IEEE 13th International Conference on Advanced Learning Technologies*, ICALT '13, pages 319–323, Washington, DC, USA, 2013. IEEE Computer Society.

[25] L. H. Werth. *Predicting student performance in a beginning computer science class*, volume 18. ACM, 1986.

[26] S. Wiedenbeck, D. Labelle, and V. N. Kain. Factors affecting course outcomes in introductory programming. In *16th Annual Workshop of the Psychology of Programming Interest Group*, pages 97–109, 2004.

[27] M. Worsley and P. Blikstein. Programming pathways: A technique for analyzing novice programmers' learning trajectories. In *Artificial intelligence in education*, pages 844–847. Springer, 2013.